ROB-GY 6413
Mechatronic Cup Tremor Compensator

Jayson Mintz (jdm9891)

Joycephine Li (jl14407)

Table of Contents

Motivation

A symptom of Parkinson’s Disease is hand tremors when the hand is at rest. This symptom can hinder daily activities in grasping objects with an example of drinking fluids out of a cup. Without tremor compensating, liquid spillage may be present when an individual with Parkinson’s Disease drinks out of a cup. Hence, the Mechatronic Cup Tremor Compensator is created to compensate for these hand tremors. This is a continuation of recreating and modifying the mechatronic cup holder presented in the paper “Tremor Compensation by Use of a Mechatronic Cup Holder” written by M. Fischer. By confirming that the movements of the physical prototype match the simulation results, the goal of recreating the mechatronic cup holder with tremor compensation is achieved.

Physical Protoype

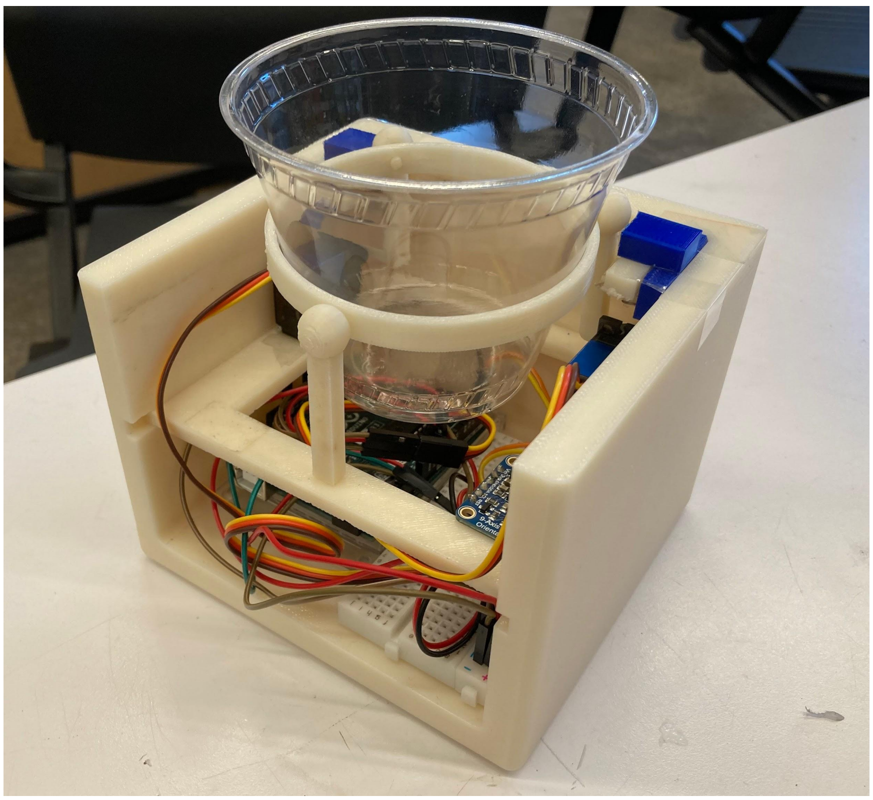

Figure 1: Mechatronic Cup Tremor Compensator

The cup holder, created by M. Fischer, consists of an ATmega2560 microcontroller, 4 accelerometers, 2 actuators, and 3D-printed model of the cup holder. With some modifications, the Mechatronic Cup Tremor Compensator currently consists of Arduino Uno microcontroller board, BNO055 9-DOF orientation sensor, 2 actuators, and 3D-printed parts used for assembling the cup holder. An Arduino Uno microcontroller is used instead of ATmega2560 microcontroller in the Arduino Mega for smaller form factor.

Bill of Materials

Table 1: Bill of Materials

<table>
<tr><th>Item</th><th>Individual Price</th><th>Number of Pieces</th><th>Tax</th><th>Total Price</th></tr>
<tr><td>Arduino Uno</td><td>$27.60</td><td>1</td><td>$0.00</td><td>$27.60</td></tr>
<tr><td>BNO055 Sensor</td><td>$34.95[1]</td><td>1</td><td rowspan="2">$5.60</td><td rowspan="2">$68.69</td></tr>
<tr><td>Micro Servo Motors</td><td>$11.95[1]</td><td>2</td></tr>
<tr><td>3D Printed Ball Joints</td><td>$0.00</td><td>1</td><td>$0.00</td><td>$0.00</td></tr>
<tr><td>3D Printed Motor Base</td><td>$0.00</td><td>1</td><td>$0.00</td><td>$0.00</td></tr>
<tr><td>3D Printed Cup Ring</td><td>$0.00</td><td>1</td><td>$0.00</td><td>$0.00</td></tr>
<tr><td>3D Printed Cup Casing</td><td>$0.00</td><td>1</td><td>$0.00</td><td>$0.00</td></tr>
<tr><td>Total (Prototype Cost):</td><td></td><td></td><td></td><td>$96.29</td></tr>
</table>

1-According to Adafruit

Mechanical Design

All the non-electrical components of the design were modeled in Fusion360 and manufactured using the 3D printers in the MakerSpace. The models included the 2-link hinged ball joints that attached to the motors, the one stationary ball joint, the ring the joints connected to that holds the cup, the base for the components to sit on, and the apparatus to hold everything which would be held by the user. One of the linked ball joints is shown below.

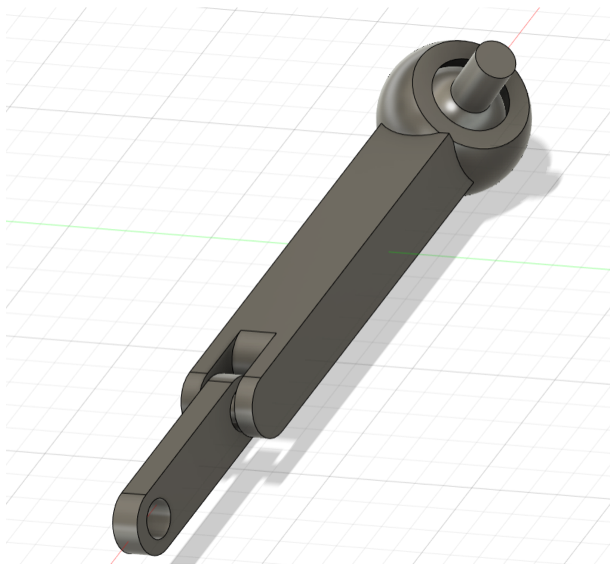

Figure 2: Left Side Ball Joint

The design used for the linkage included a print-in-place ball joint and hinge. By printing the entire linkage as one component, it reduced the amount of assembly needed for the system. The third ball joint which remained stationary followed the same design as the ball joint above, just without the hinged link. The ring that the joints connect to is shown below.

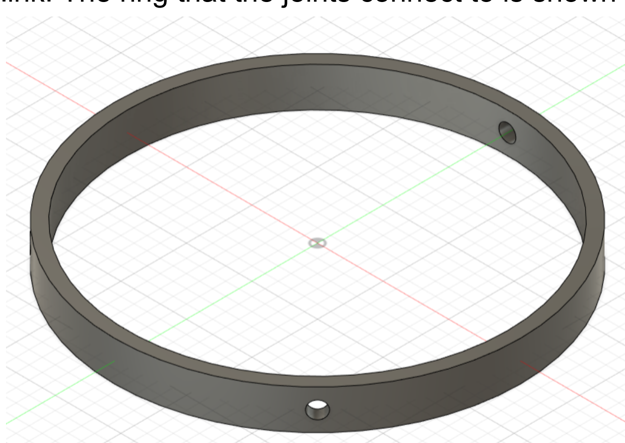

Figure 3: System Ring

The peg on each joint plugs into the holes in the ring. The hole in the back is centered and the holes in the front are an equal distance from the back hole. The base of the model is shown below.

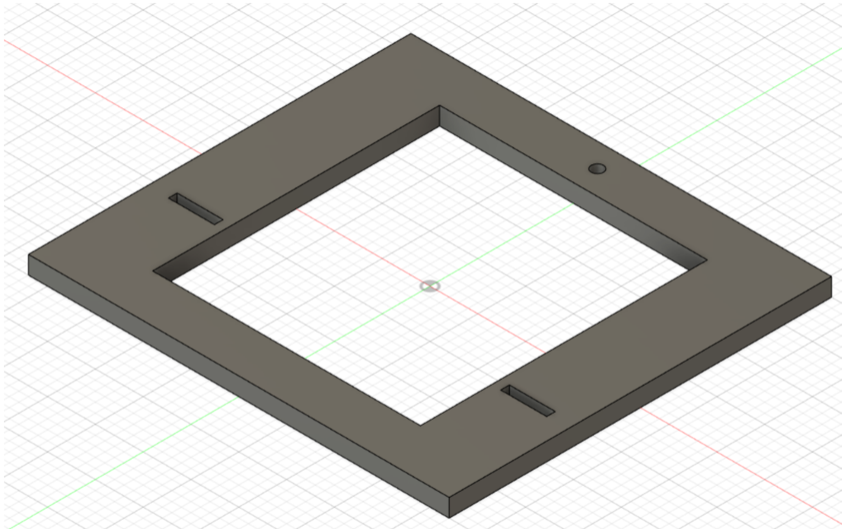

Figure 4: System Base

There are three holes in the base. The circular hole in the back is where the stationary joint connects, and the rectangular holes allow for the motors to sit flush against the base. The base slides into the system apparatus shown below.

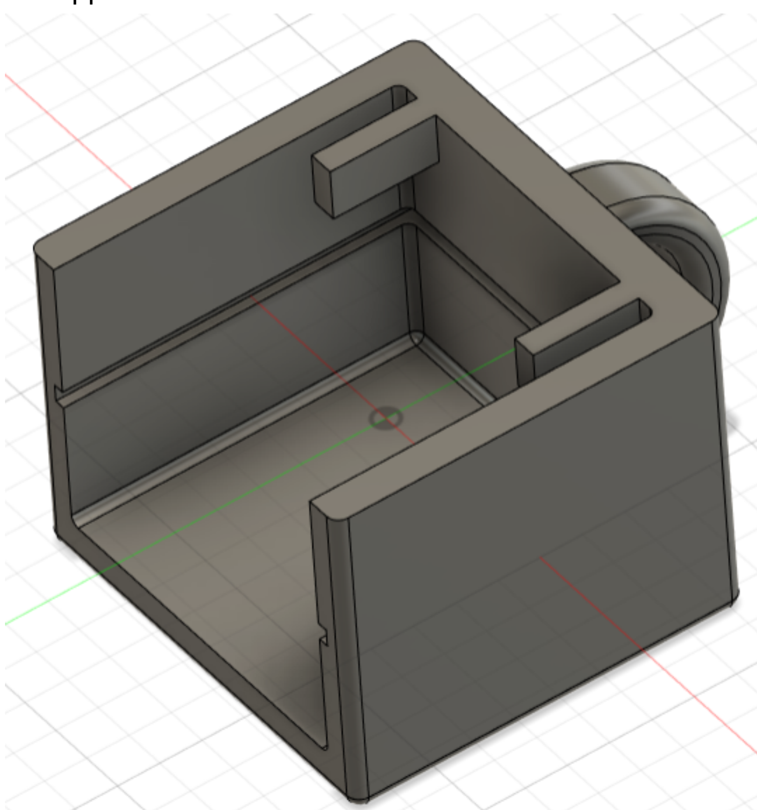

Figure 5: System Apparatus

This component was designed last as the base was used for initial testing. It was determined that within the time constraint of the project that designing a holder around the base so it could slide in would be the most efficient. The 2 tabs at the top were added to help keep the longer links of each chain upright as without them, the system would often tilt to the left or the right due to the nature of the ball joints. After printing this component, the tabs weren't wide enough in order to properly support the links, so the simple component shown below was printed to sit in the space between the wall and tab to help support the links.

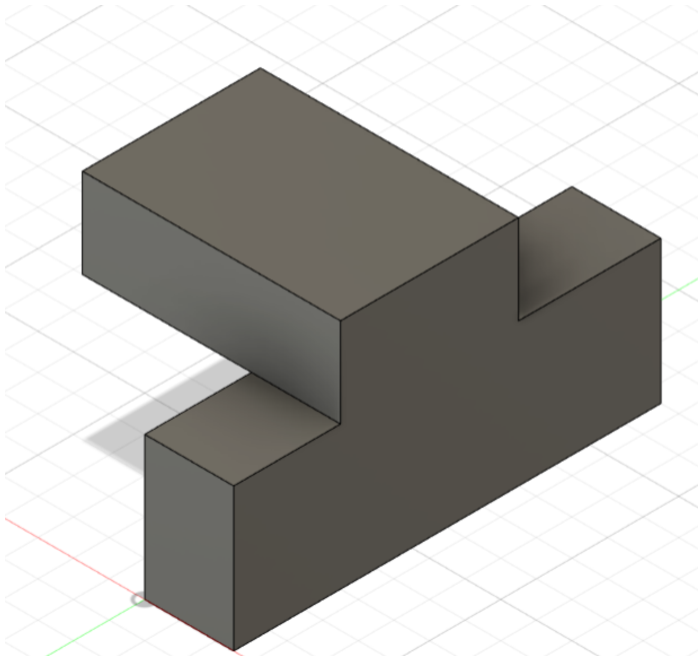

Figure 6: Linkage Support

Electrical Circuits

Below is the overall schematic diagram of the mechatronic cup holder.

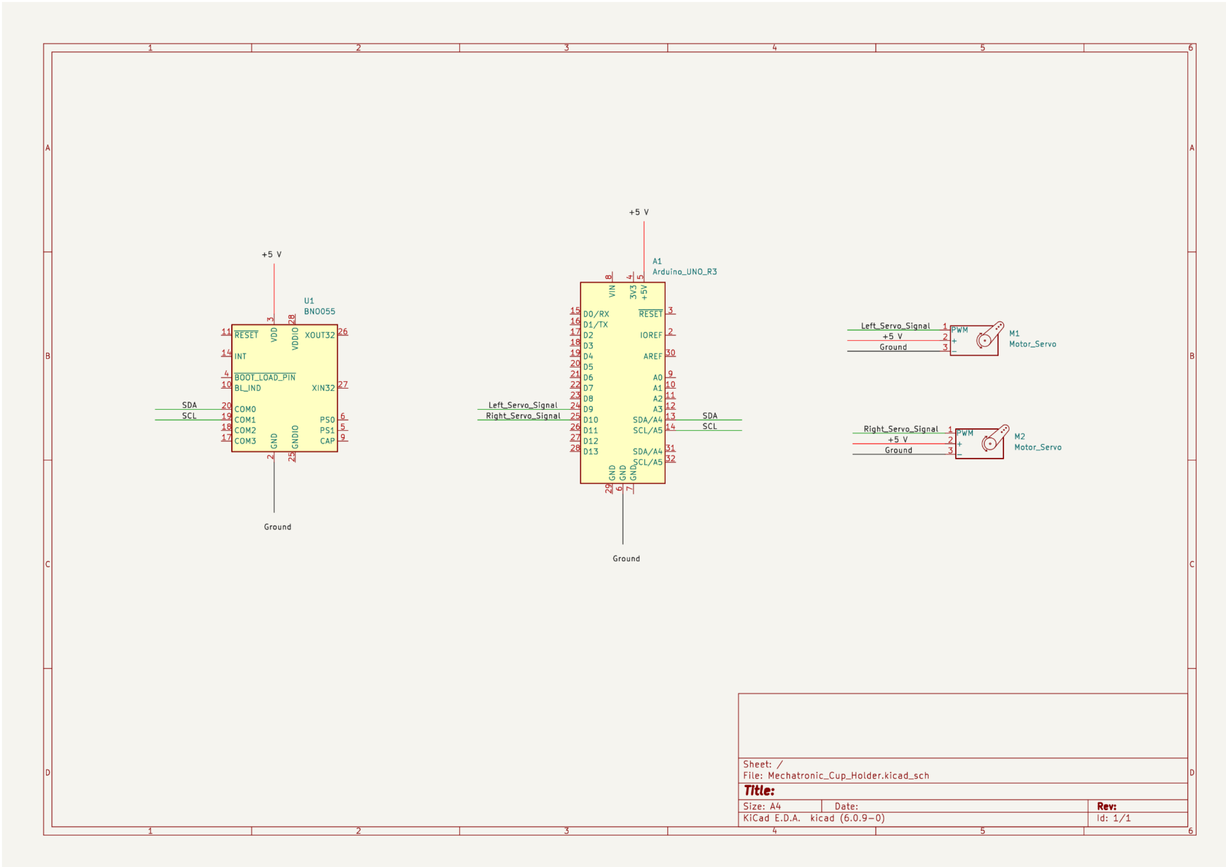

Figure 7: Overall Wiring Schematic

BNO055 9-DOF Orientation Sensor

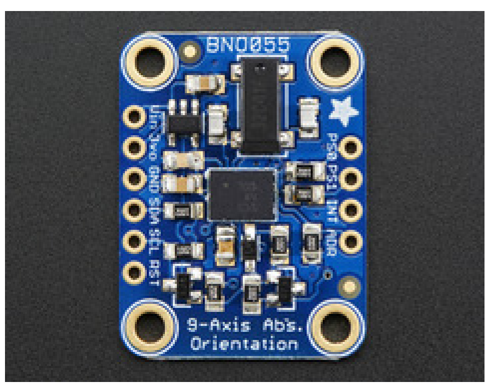

Figure 8: BNO055 9-DOF Orientation Sensor

The BNO055 Orientation Sensor consists of an accelerometer, magnetometer, and gyroscope that can output vectors of acceleration (both gravitational and linear), angular velocity, and magnetic field strength. This sensor also has fusion mode, where sensor fusion computations within an ARM Cortex-M0 processor can output vectors of relative and absolute orientation (in Euler angles and quaternions) using accelerometer, gyroscope, and magnetometer data. The orientation sensor operates at +3.3 V and uses I2C connection, so the sensor utilizes A4 (SDA) and A5 (SCL) pins of the Arduino Uno.

M. Fischer originally used 4 accelerometers to calculate the current position and orientation of the cup holder. Instead of computing the current position and orientation based on accelerometer values, the BNO055 sensor does the calculations (outputting Euler angles), which saves lines of code. For this instance, angular velocity (of units rads/s) and linear acceleration (of units m/s^2) in x and y axes were used as inputs for the sloshing model explained in the code section.

Micro Servo Motors

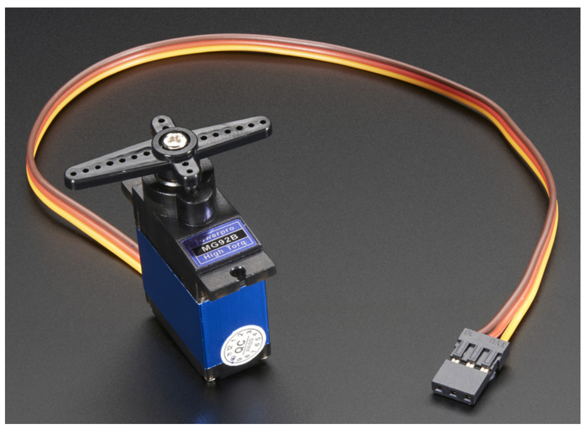

Figure 9: Micro Servo Motor

The micro servo motors operate at +5.0-6.6 V power with +5 V logic. Just like standard servo motors, these micro servo motors consist of three connections: Signal, +5 V, and Ground. The reason for selecting this motor is the stall torque of 3.5 kg/cm powered at +6.0 V matches the torque of the motor M. Fischer used in the mechatronic cup holder. These servo motors are also compatible with motor controllers and servo libraries and codes, including the Arduino builtin library.

The actuators adjust from the current pose and orientation to the desired orientation based on the angles of the liquid. To adjust and determine angles of the motor, inverse kinematics is calculated within the code after finding the current pose and orientation. The angles provided in the code were converted from radians to degrees because the parameters of the builtin library are in units of degrees. The signal pins are connected to the PWN pins 9 and 10 and powered by a +5 V given in Arduino. Ground is connected to the ground of the Arduino.

Code

The following flowchart demonstrates our coding methods for this project:

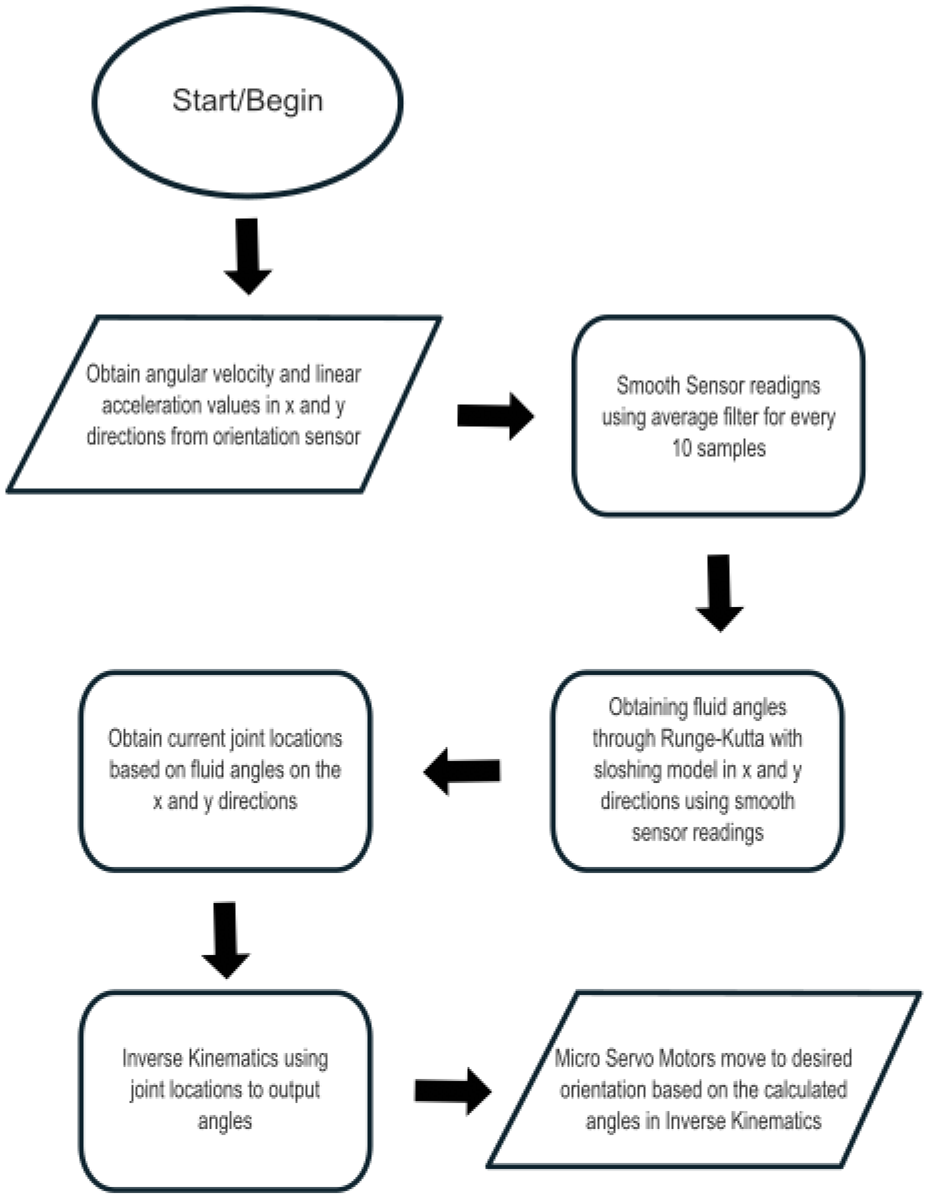

Figure 10: Flowchart Diagram

The algorithm written in the Arduino IDE was similar to the paper, where M. Fischer calculated the fluid angle, the current pose and orientation of the cup holder, and inverse kinematics. After Arduino Uno was able to detect the BNO055 orientation sensor, the sensor outputted a vector of raw sensor values of angular velocity (in units of rad/s) and linear acceleration (in units of m/s^2). Because water movement in a container is considered pendulum-type with inclined path and rotational motion (2-dimensions), only values in x and y axes were retrieved. Then, an average of 10 readings (of angular velocities and linear accelerations in both x and y axes) was taken for smoother readings due to the amount of noise in the sensor.

M. Fischer's paper stated discretization would need to be applied to the sloshing model shown in Equation 1.

$$\ddot{\theta}_x = -\frac{c}{m}\cdot\left(\dot{\theta}_x - \dot{\eta}_x\right) - \frac{g}{l}\cdot\theta_x - \frac{1}{l}\cdot a_x.$$

Equation 1: Sloshing Model in x-direction

Hence, translating the equation into a differential equation would result in equation 2. To solve Equation 2, M. Fischer used a two-step procedure. Instead of a two-step procedure, Runge-Kutta method is used to obtain updated accurate readings. With the self-defined Runge-Kutta function, Equation 2 is considered for both x and y axes.

$$\begin{pmatrix} \dot{z}_1 \\ \dot{z}_2 \end{pmatrix} = \begin{pmatrix} -\frac{c}{m} & -\frac{g}{l} \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} \frac{c}{m}\cdot\dot{\eta} + \frac{1}{l}\cdot {}^{world}a \\ 0 \end{pmatrix}, \quad (14)$$

Equation 2: Sloshing ODE

where *c* is the viscosity of the fluid and friction between fluid and cup, m is the mass of the liquid, *g* is the gravitational acceleration, ℓ is the length of the pendulum from eigenfrequency of liquid, the derivative of η is the angular rate of the cup holder, $world_a$ is the linear acceleration, z_1 is angular velocity of the fluid, and z_2 is the angle of the fluid. *c*, m, *g*, and ℓ were given in the paper as the following, respectively: 0.54 Ns/kg, 0.2 kg, 9.81 m/s^2, and 0.092 m.

Using Runge-Kutta method outputted the updated angular velocities and linear accelerations. The updated values consisted of conditions that determine the fluid angles in x and y axes. These fluid angles were used to find the joint locations of the cup holder. Due to difficulties in forming matrices in Arduino, the statements under *jointLocations()* function were found through calculations in MATLAB using symbolic math. After finding the joint locations, inverse kinematics was used to find the angles of the motors (converted from radians to degrees) to reach the desired orientation and pose. The code utilized in MATLAB for the simulation project was translated over to Arduino's syntax, but otherwise the kinematic functions remained unchanged.

Conclusion

The ball joints of the physical Mechatronic Cup Tremor Compensator operated the same as the results from the simulation, which was done earlier as the first half of the project. The sloshing was also controlled based on the demonstration video. Overall, the mechatronic cup was able to compensate for hand tremors at rest to prevent spillage. Furthermore, an individual can drink the liquid out of the cup as the system can differentiate when the individual is drinking fluid out of the cup or grasping the cup.

Despite the prototype working, there are some improvements that can be done to make the design more convenient. The casing of the cup can be lighter as the overall cup design was heavier than a mug. Another improvement can be using different material for the ball joint for durability purposes in considering the load of the motor. A future step of the project would be waterproofing the electrical components in case of water droplets. Furthermore, we can observe how the overall system can withstand more water in the cup as we can only confirm with a small amount of water.

References

[1] Bosch Sensortec. “BNO055 Intelligent 9-axis absolute orientation sensor.” *Adafruit*, 15 October 2013, https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf. Accessed 27 November 2023.

[2] “Fourth Order Runge-Kutta.” *LPSA (Linear Physical Systems Analysis)*, https://lpsa.swarthmore.edu/NumInt/NumIntFourth.html. Accessed 13 December 2023.

[3] “Micro Servo - High Powered, High Torque Metal Gear [TowerPro MG92B] : ID 2307 : $11.95.” *Adafruit*, https://www.adafruit.com/product/2307#description. Accessed 8 December 2023.

[4] Townsend, Kevin. “Arduino Code | Adafruit BNO055 Absolute Orientation Sensor.” *Adafruit*, 22 April 2015, https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/arduino-code. Accessed 27 November 2023.

Appendix

Code

```
//libraries for sensor communication
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <Servo.h>

Adafruit_BNO055 bno = Adafruit_BNO055(55);

// BNO055 Vectors
imu::Vector<3> w;
imu::Vector<3> a;

// Define constants
const float g = 9.81;   // Acceleration due to gravity (m/s^2)
const float L = 0.092;  // Length of the pendulum (m)
const float c = 0.54;   // coefficient
const float m = 0.2;    // Mass of the liquid (kg)


unsigned long previousMillis = 0;
const long interval = 10;

// Filter constants
const int numReadings = 10;  // Number of readings to average
float omegaReadingsX[numReadings];
float aReadingsX[numReadings];
float omegaReadingsY[numReadings];
float aReadingsY[numReadings];
int readingIndex = 0;

```

```
// Initial conditions
float thetaX = 0.0;  // Initial angular displacement (rad)
float omegaX = 0.0;  // Initial angular velocity (rad/s)
float thetaY = 0.0;  // Initial angular displacement (rad)
float omegaY = 0.0;  // Initial angular velocity (rad/s)
float fluidX = 0.0;  // Initial fluid x angle (rad)
float fluidY = 0.0;  // Initial fluid y angle (rad)

// Joint Locations
float leftX;
float leftY;
float rightX;
float rightY;

// Servos
Servo left_servo;
Servo right_servo;
float leftAngle;
float rightAngle;

```

```
void setup() {
  // Initialize sensor and attach servos
  Serial.begin(9600);
  left_servo.attach(9);
  right_servo.attach(10);
  /* Initialise the sensor */
  if (!bno.begin()) {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while (1)
      ;
  }
  delay(1000);
  bno.setExtCrystalUse(true);
  // Initialize filter arrays
  for (int i = 0; i < numReadings; i++) {
    omegaReadingsX[i] = 0.0;
    aReadingsX[i] = 0.0;
    omegaReadingsY[i] = 0.0;
    aReadingsY[i] = 0.0;
  }
}

```

```
void loop() {
  // Get sensor readings (replace with your sensor reading functions)
  sensors_event_t event;
  bno.getEvent(&event);
  a = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
  w = bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
  float rawOmegaX = w.x();
  float rawAX = a.x();
  float rawOmegaY = w.y();
  float rawAY = a.y();

  // Calculate time elapsed since the last iteration
  unsigned long currentMillis = millis();
  unsigned long elapsedTime = currentMillis - previousMillis;


  // Smooth the sensor readings using a moving average filter
  float omegaX = smoothSensorData(rawOmegaX, omegaReadingsX);
  float aX = smoothSensorData(rawAX, aReadingsX);
  float omegaY = smoothSensorData(rawOmegaY, omegaReadingsY);
  float aY = smoothSensorData(rawAY, aReadingsY);

  // Check if it's time to perform the next iteration
  if (elapsedTime >= interval) {
    // Save the current time for the next iteration
    previousMillis = currentMillis;

    // Runge-Kutta 4 method
    rungeKutta4(omegaX, aX, omegaY, aY);

  }

```

```
  if (thetaX * 180 / 3.141592 > 4) {
    fluidX = 4 * 3.141592 / 180;
  } else if (thetaX * 180 / 3.141592 < -2) {
    fluidX = -2 * 3.141592 / 180;
  } else {
    fluidX = thetaX;
  }

  if (thetaY * 180 / 3.141592 > 8) {
    fluidY = 8 * 3.141592 / 180;
  } else if (thetaY * 180 / 3.141592 < -8) {
    fluidY = -8 * 3.141592 / 180;
  } else {
    fluidY = thetaY;
  }

  jointLocations();
  IK();

  Serial.print("X: ");
  Serial.print(leftAngle);
  Serial.print("\tY: ");
  Serial.print(rightAngle - 3.141592);
  Serial.println("");
  left_servo.write(leftAngle * 180 / 3.141592 + 90);
  right_servo.write(rightAngle * 180 / 3.141592 - 90);
  delay(50);
}
```

```
float smoothSensorData(float rawValue, float readings[]) {
  // Apply a simple moving average filter
  // Shift the readings and add the new value
  float sum = 0.0;
  readings[readingIndex] = rawValue;

  for (int i = 0; i < numReadings; i++) {
    sum += readings[i];
  }

  readingIndex = (readingIndex + 1) % numReadings;

  // Return the average
  return sum / numReadings;
}

```

```
void rungeKutta4(float omegaX, float aX, float omegaY, float aY) {
  // Runge-Kutta 4 method
  float dt = interval / 1000.0;  // Convert interval to seconds
  float kx1, kx2, kx3, kx4;
  float ky1, ky2, ky3, ky4;

  kx1 = sloshingOdeX(omegaX, aX);
  kx2 = sloshingOdeX(omegaX, aX + dt / 2 * kx1);
  kx3 = sloshingOdeX(omegaX, aX + dt / 2 * kx2);
  kx4 = sloshingOdeX(omegaX, aX + dt * kx3);

  ky1 = sloshingOdeY(omegaY, aY);
  ky2 = sloshingOdeY(omegaY, aY + dt / 2 * ky1);
  ky3 = sloshingOdeY(omegaY, aY + dt / 2 * ky2);
  ky4 = sloshingOdeY(omegaY, aY + dt * ky3);

  // Update variables
  thetaX = thetaX + dt * (kx1 + 2 * kx2 + 2 * kx3 + kx4) / 6;
  omegaX = omegaX + dt * (kx1 + 2 * kx2 + 2 * kx3 + kx4) / 6;
  thetaY = thetaY + dt * (ky1 + 2 * ky2 + 2 * ky3 + ky4) / 6;
  omegaY = omegaY + dt * (ky1 + 2 * ky2 + 2 * ky3 + ky4) / 6;
}

```

```
float sloshingOdeX(float omega_ext, float a_ext) {
  // ODE function for sloshing motion
  return -g / L * thetaX - c / m * omegaX + c / m * omega_ext + a_ext / L;
}
float sloshingOdeY(float omega_ext, float a_ext) {
  // ODE function for sloshing motion
  return -g / L * thetaY - c / m * omegaY + c / m * omega_ext + a_ext / L;
}
float jointLocations() {
  float cup_z = 1.25;
  float ring_rad = 1.25;
  float ring_x = sqrt(sq(ring_rad + 0.5) / 2);
  float ring_y = sqrt(sq(ring_rad + 0.5) / 2);

  leftX = ring_x - ring_x * cos(fluidY) + 0.5;
  leftY = cup_z + sin(fluidX) * (ring_y + ring_rad) - ring_x * cos(fluidX) * sin(fluidY);
  rightX = ring_x * cos(fluidY) - ring_x - 0.5;
  rightY = cup_z + sin(fluidX) * (ring_y + ring_rad) + ring_x * cos(fluidX) * sin(fluidY);
}

```

```
float IK() {
  float a1 = 0.5;
  float a2 = 1.35;

  double q2l_num = sq(leftX) + sq(leftY) - sq(a1) - sq(a2);
  double q2l_den = 2 * a1 * a2;

  // Left
  double q2_l = acos(q2l_num / q2l_den);
  double q1_l = atan(leftY / leftX) - atan((a2 * sin(q2_l)) / (a1 + a2 * cos(q2_l)));
  if (q1_l > 3.141592 / 4.0) {
    q1_l = q1_l - 3.141592;
  }
  leftAngle = q1_l;

  // Right
  double q2r_num = sq(rightX) + sq(rightY) - sq(a1) - sq(a2);
  double q2r_den = 2 * a1 * a2;

  double q2_r = -acos(q2r_num / q2r_den);
  double q1_r = 3.141592 - atan(-rightY / rightX) - atan((a2 * sin(q2_r)) / (a1 + a2 * cos(q2_r)));
  if (q1_r < 3.141592 / 4.0) {
    q1_r = q1_r + 3.141592;
  }
  rightAngle = q1_r;
}
```